RE PAGE	2.	3. Recipient's Accession No
4. Title and Subtitle **Real-Time Reconfiguration Study (Final Report)**		5. Report Date **August 1994**
		6.
7. Author(s) **Daniel Searles**		8. Performing Organization Rept. No.
9. Performing Organization Name and Address **Loral Federal Systems** **Manassas, Virginia**		10. Project/Task/Work Unit No. N/A
		11. Contract(C) or Grant(G) No. (C)N00014-91-C-0129 (G)
12. Sponsering Organization Name and Address **Office of Naval Research**		13. Type of Report & Period Covered
		14.

15. Supplementary Notes

16. Abstract (Limit: 200 words)

17. Document Analysis a. Descriptors

None

b. Identifiers/Open-Ended Terms

None

c. COSATI Field/Group

None

18. Availibility Statement	19. Security Class (This Report) Unclassified	21. No. of Pages 23
	20. Security Class (This Page) Unclassified	22. Price

Real-Time Reconfiguration Study (Final Report)

Prepared by

Daniel Searles

Loral Federal Systems (formerly IBM Federal Systems)
Manassas, Virginia

August 1994

Contract:

N00014-91-C-0129

Prepared for

Office of Naval Research

Trademarks

Interleaf	Interleaf, Inc.
pSOS+, pROBE+, pHILE+, and pNA+	Integrated Systems Inc.
Sun Microsystems, Sun Workstations, Open Windows, Sun View, Sun-2, Sun-3, Sun-4, Sun-386i, SPARC, SPARCSTation, SUN OS	Sun Microsystems Incorporated
UNIX	AT&T Bell Laboratories

Abstract

The objective of the real-time reconfiguration study was to implement a physical test bed for testing and evaluating a suite of reconfiguration algorithms. The software architecture was composed of an operating system which supported real-time programming concepts, a set of model tasks, control software, and the suite of reconfiguration procedures. Rate Monotonic scheduling was used to schedule the execution of the task set, as well as for testing the schedulability of new configurations. Of primary concern was the execution time of the reconfiguration algorithms. The general problem of reconfiguration is similar to the bin packing problem and is NP–Complete. This suite implements a concept of "disturbance", a cost associated with moving a task from one CPU to another, to reduce execution time. The hardware platform was composed of a VME chassis populated with 5 Motorola 68030 CPUs.

Contents

Figures

Acknowledgments

The initial implementations of the RECONF, RFFD, and MFFD algorithms were done by Gary Cornwell.

The burning of ROM chips and the construction of the VME chassis were done by Mark Jones.

List of Symbols, Abbreviations and Acronyms

CPU	Central Processing Unit
FFD	First Fit Decreasing
ONR	Office of Naval Research
OS	Operating System
RMS	Rate Monotonic Scheduling

Summary

This report documents the implementation and evaluation of the reconfiguration algorithms developed for IBM by Dr. John Lehoczky and Lui Sha of Logic Associates. The algorithms were implemented on a standalone system for testing and debugging. Then the algorithms, along with a model task set and control software, were transported and integrated on a test bed composed of a VME chassis and 5 Motorola 68030 CPUs. The net result verified that the test bed performed as predicted by the algorithms as long as the set of tasks were independent.

Introduction

This report documents the effort to implement a set of reconfiguration algorithms developed for IBM by Dr. John Lehoczky and Lui Sha and evaluate their performance. The reconfiguration problem is the problem of finding a way to place a set of modes (a mode is a set of processes) into a set of CPUs without exceeding CPU, memory, or other system constraints such that all hard–deadline response requirements are met. Usually a reconfiguration problem will start with a working system (mode to CPU mapping), and a CPU is removed or disabled, causing a need to redistribute the modes among the remaining CPUs. This type of reconfiguration has potential use in systems where it is critical that processing continue in the presence of failed or disabled CPU resources in such a way that critical system response requirements are satisfied..

In Phase I of the study IBM established the process model for a sample system problem. This included a definition of how many concurrent system modes would be active, how many processes comprised each mode, whether processes could be co–resident with certain other processes, what the functional criticality of the modes were, etc. Also, included in Phase I was the design of a test bed. During Phase II, IBM was to implement the algorithms on a physical network/processor test bed, to evaluate performance of a live system, relative to the theoretical performance predicted by the algorithms.

The algorithm set consists of five algorithms: RECONF, RFFD, MFFD, CLUSTER, and SCHED. RECONF is the controller. It initializes the system, accepts user inputs, and calls the other algorithms as needed. RFFD stands for Reverse First Fit Decreasing, and is responsible for computing which modes may be moved during a reconfiguration attempt. MFFD stands for Modified First Fit Decreasing. This algorithm places movable modes into CPUs according to their characteristics and resources. CLUSTER is used if the results of MFFD do not satisfy interprocess communications constraints. SCHED, used when processes are being placed into CPUs, is used to determine if the set of processes assigned to a CPU will all meet their deadlines under the Rate Monotonic Scheduling paradigm. Other constraints consist of CPU utilization, memory utilization, pseudo resources, communications channel bandwidth and disturbance limitations.

In the mode/process model used on the test bed, modes are composed of one or more processes, and the processes may pass data among themselves (via TCP/IP sockets). Rate Monotonic Scheduling, however, requires that the set of processes be independent (meaning no process can effect the execution of another process except by higher priority preemption) before it can guarantee that all the processes will meet their deadlines. If the socket calls are made non–blocking so that the processes are independent, issues regarding data integrity and synchronization are raised. For this report no interprocess communication requirements were defined in any of the test cases.

Since the problem of reconfiguration is NP–Complete (similar to bin packing), exhaustive exploration of all possible combinations becomes computationally prohibitive at some point, especially if encountered in a real–time environment. The algorithms developed do not attempt to test all possible combinations, but rather try to satisfy a set of criteria. The criteria used by these algorithms is based on a concept of "disturbance". Each mode is assigned a disturbance value by the designers of the system which approximates the cost of suspending a mode, moving it to another CPU and reactivating it. The concept is extended slightly to handle non–linear cases, such as when the cost of moving a pair of modes is more than the sum of the costs of the individuals. The algorithms go one step further, and also try to minimize future disturbance. Future disturbance is the theoretical cost of reconfiguring a system again, due to another failure. Since not all possible combinations are checked, the algorithms may fail to generate any configuration, even though a valid configuration exists, or they may not compute the best configuration possible. But the approach taken by the algorithms attempt to do the best job possible considering the constraints.

Overview

Hardware/Software Environment

The real–time reconfiguration study was supported by a number of platforms and environments during its development. The target development and test platform consisted of the following components:

- SUN 3/470, with 32 Mb memory. Sun OS 4.0.3.
- Five Motorola MVME–1475A–1 or MVME--1475A–2 CPU boards mounted in a VME chassis.
- Five VME transfer cards supporting Ethernet interfaces.
- Ethernet to connect Sun development/Test platform with CPU boards.

The five CPU boards and transfer cards were installed in a single VME chassis. Figure 1 on page 3 show how the boards were arranged. Note that the speed of the CPUs on the 1475A–1 and 1475A–2 are not the same, but this was accounted for in the tests. The reconfiguration algorithms require a set of identical CPUs.

For the operating system on the CPU boards, pSOS ™ from Integrated Systems, Inc. was used. It supports priority tasking with preemption, remote debug, and TCP/IP sockets. One of the major design goals was to restrict interprocess communication to TCP/IP sockets, and avoid communications via the VME back plane. So, although the initial test bed consisted of a single VME chassis, the VME back plane was not utilized for interprocess communication purposes.

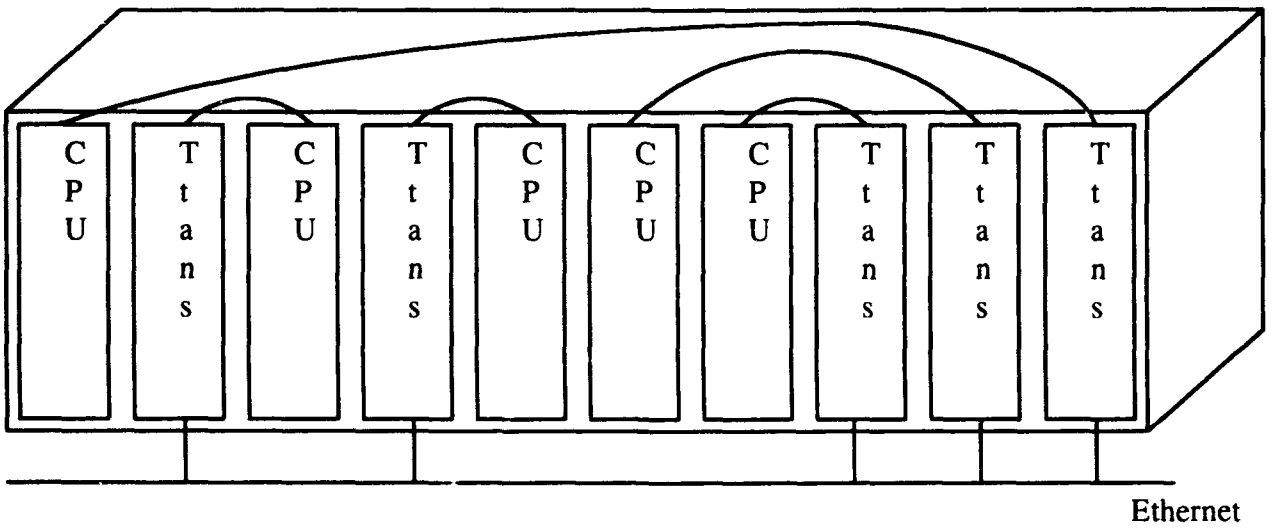


Figure 1 MVME–147 CPU/Transfer board arrangement in VME chassis

Process Model

Under the pSOS operating system there is a 'C' function designated as the "root" which is initialized to run when a program is down loaded to a CPU board. This task is also initialized to run at the highest priority available. In the implementation of the reconfiguration algorithms, the root task is used as a task controller

for the other processes assigned to that particular CPU. One CPU is designated the master which also has the responsibility for polling the other CPUs for detecting CPU failure, and computing and transferring reconfiguration data. The other CPUs are thus considered slaves. One of the responsibilities for the root process in each CPU is to initiate the complete set of processes. Each process then determines if it should be executing or not based on which CPU it is in. In this implementation a single flexible process model is used to simulate process execution. The code for each process is identical, but execution is controlled by a table of parameters. This method allows a process to initiate data transfers, receive data transfers or both or neither, to/from any other process in the system. By consulting the global configuration data, each process can determine if the process with which it is to interact is located on the same CPU or not and to act appropriately. The following design segment is for the root controlling task.

```
/* Design for the Root control process */
Root(MyCPU)
Begin
  Initialize trace tables, local variables, the network interface
  Read initial CPU configuration data
  Create set of generic processes

  Open sockets to root processes in other CPUs

  Loop forever
    If this root process is in the master CPU
    Then
      Send 'query' messages to other CPUs
      Receive responses from other CPUs
      If new CPU failure detected
      Then
        Suspend all generic processes
        Send 'starting reconfiguration' message to other CPUs
        Execute reconfiguration algorithms
        Send new configuration data to other CPUs
        Restart and resume generic processes
      End if
      Sleep 10 seconds

    Else  /* This is a root process in a slave CPU */

      Wait for message from root master CPU
      If
        'query' message received
      Then
        Send 'Query response'
      Else
```

```
            /* 'starting reconfiguration' message received */
          Suspend all generic processes
          Receive new configuration data
          Restart and resume generic processes
        End if
      End if
    End loop forever
End Root
```

Figure 2 Root Process Design

The following design segment is for the generic process model. The calculation of the deadline is computed as process period plus the previous deadline. This method of computing a deadline maintains an environment where Rate Monotonic Scheduling can be used. At the bottom of the cycle of the model process, the current time is queried. When current time is before the computed deadline time, the task is suspended until the deadline to maintain its period. If the process has run beyond its deadline, a deadline error is recorded, and the task is not suspended to minimize the delay until the start of the next cycle. This is one simple method that maintains a Rate Monotonic Scheduling Structure. The pSOS operating system supports tasks with priorities and task preemption.

```
/* Standard multi-purpose Client/Server task model */
Generic_Process(Procid)
Begin
  Initialize counters, trace data, and other local variables.
  Block here if this process is not assigned to this CPU.
  Compute initial deadline

  Loop forever
    If CPU configuration has changed
    Then
      Close open sockets
      If input is from a remote CPU
      Then
        Open socket to remote CPU
      End if
      If output is to a remote CPU
      Then
        Open socket to remote CPU
      End if
    End if

    If input is from a remote CPU
    Then
      Read data from socket
    End if

    Perform CPU intensive processing, to simulate expected CPU usage.

    If output is to a remote CPU
    Then
      Write data to socket
    End if
```

```
    Compare deadline time to current time
    If deadline passed
    Then
      Record deadline passed Error
    End If

    Compute new deadline time
    If new deadline time has not been passed
    Then
      Sleep until deadline time
    End If

    Block here if this process is not assigned to this CPU
  End Loop forever
End Generic_process
```

Figure 3 Model Process Design

Program Notes

The above designs are only one of many possible designs. Several factors are assumed or over looked in favor of simplifying the implementation. Some, but not all of the factors include:

- No accounting for process state data. When a process is moved from one CPU to another, it will lose its state data unless some extra action is taken.
- Reconfiguration stops and restarts all generic processes, even those unaffected by the reconfiguration. Ideally, only those processes actually relocated should be effected.
- At most one possible input and one possible output per process cycle. The parameters which control input/output allow for skipping cycles to allow processes with different periods to transfer data (as long as one period is a multiple of the other). Note no input/output requirements were defined to allow the process set to remain independent.
- No CPU time is used to move or copy data into/out of a process when its input/output is to a process in the same CPU.
- Each process uses it rated percent CPU each cycle. The CPU used by a process is really the sum of the `'CPU intensive processing'` phase plus the time spent getting or sending input or output data. Since the same number is used to control the length of the `'CPU intensive processing'` phase as which defined the CPU usage of the process, the time spent getting or sending input or output data is not accounted for.

For testing, a data file containing the initial configuration is the source of input for the reconfiguration program. Compiler flags allow the development of slightly different versions of the program. One version allows the name of the input data file to be a command line parameter. Another, planned for use on the test bed, gets its input data from a 'C' code program fragment. This is required because file I/O is not supported in our test bed implementation. The compiler flags also allow for various levels of debug capability by including or excluding 'printf()' statements. The test bed implementation is not expected to support 'printf()' capability, so those functions are also excluded in the test bed version. To monitor the execution of the processes down loaded to the CPU modules, the pSOS interactive debugger is used. Every attempt was made

to minimize overhead associated with the monitoring of the execution of the processes, such as storing trace data in memory while the tests were executing and reviewing them afterwards.

The root process, and a system network process have the highest priority on the test bed. It is unknown what the exact nature of the network process is other than it processes incoming and outgoing network packets. Its operation is considered overhead. The root process also has a priority higher than any of the generic processes. The root process, however, is not following the RMS rule where its priority and period should be inversely proportional. Currently it polls the other CPUs once every 10 seconds. During normal operation of a test, its processing is also considered overhead. During reconfiguration, which may consume a substantial amount of CPU, the reconfiguration algorithms run at the root level of priority. But, since all processes are suspended, this is not a concern. If, however, the processing of the reconfiguration data is improved to the point where only processes affected by a change are disrupted, this design will have to be changed to accommodate this processing. Reconfiguration processing is acyclic by nature, but, in order to not disrupt the currently executing processes in the same CPU, it should be run at a very low priority.

Probably the biggest departure from real world systems is the problem caused by real world sets of processes not being truly independent. As an initial step it was appropiate to assume independence as this will provide a baseline against which future, more complex studies may be measured. Even when the process set is made independent by using non–blocking reads/writes, new questions concerning data integrity and synchronization occur. Typically system designers "hardcode" a time line in their systems so they can verify data flows at the appropriate times and rates. These systems rarely account for changes in CPU resources or even have the ability to relocate processes. Investigation into the problems and solutions of data integrity and synchronization with independent processes was beyond the scope of this study.

Results and Discussion

Descriptions and Test Cases 1–3

The following figure explains the notation used to represent CPUs, modes, and processes.

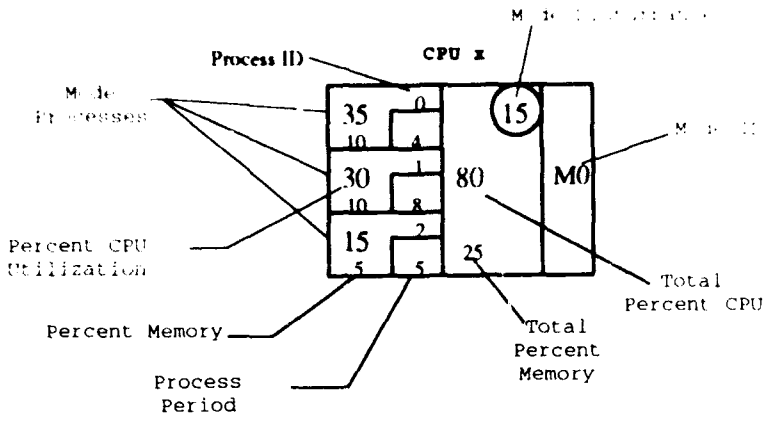


Figure 4 Description of a mode description block.

A system is composed of a set of CPUs and a set of modes. A mode is comprised of a set of related processes (one or more). The processes are depicted on the left side of the mode description block. Each compartment represents a separate process and includes data on the processe's CPU usage, memory requirements, period of execution and a unique id. The middle portion shows total CPU/memory utilization, and the disturbance factor associated with the mode. The right edge shows the mode id. The algorithm set supports pair–wise and triple–wise disturbance factors among modes but no such factors were defined for tests in this report.

The first example (figure 5 on page 9) shows the standard test case consisting of 5 CPUs, 8 modes and 19 processes. The fifth CPU is empty and considered a spare. For cases where only one of the other CPUs (except CPU 0) fail, the modes and processes of the failed CPU are merely relocated to the spare. In the test bed system CPU 0 is used to detect failures in the other CPUs and control reconfiguration and is not allowed to fail. The ratio at the bottom of each CPU is the total CPU/total memory utilization for that CPU.

The first set of reconfiguration examples are generated from the standard test case by failing CPU 1 and CPU 4 (the spare), then CPU 2 and the spare, and then CPU 3 and the spare. Each of these cases forces the processes in the failed CPUs to be distributed to other still functioning CPUs. The reconfiguration algorithms are run and they attempt to minimize total system disturbance, but also respect CPU resources and the Rate Monotonic schedulability of the resulting task sets.

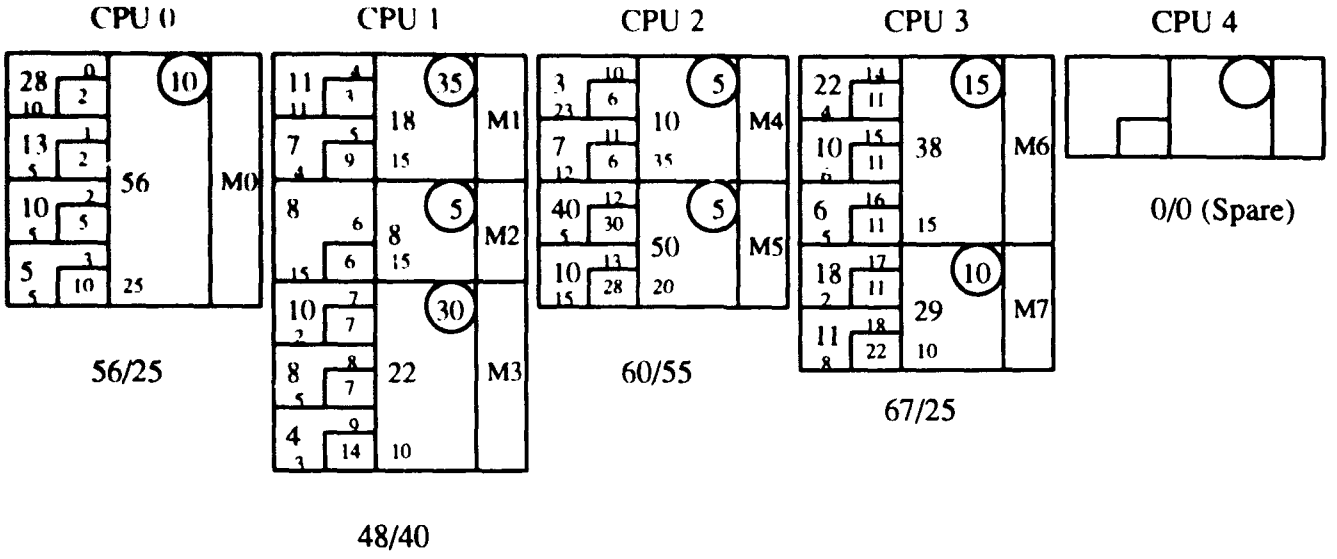


Figure 5. Standard initial system configuration for test cases 1–3.

Test case 1, CPUs 3 and 4 are marked as failed, and the resulting configuration is shown in figure 6 on page 9.

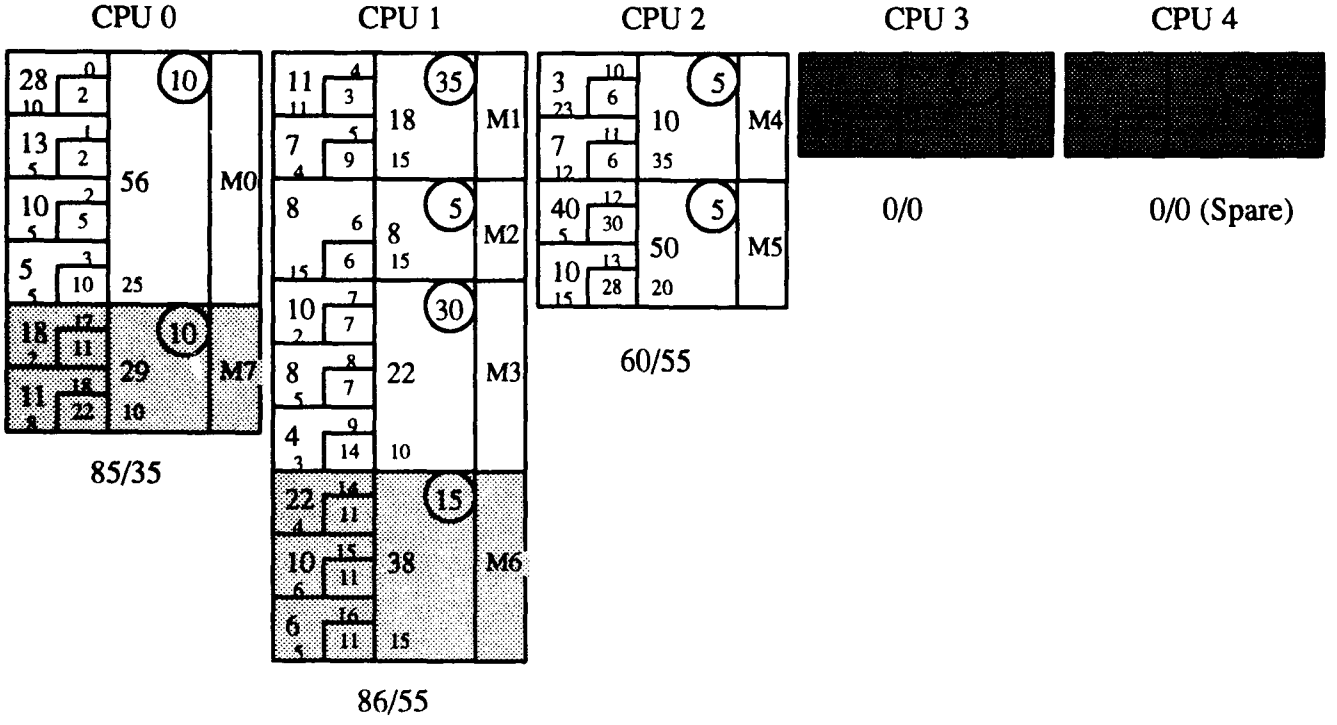


Figure 6 Reconfiguration Case 1: After

Test case 2, CPUs 2 and 4 are marked as failed, and the resulting configuration is shown in figure 7 on page 10. In this case the two processes of mode 5 are split apart and placed in different CPUs.

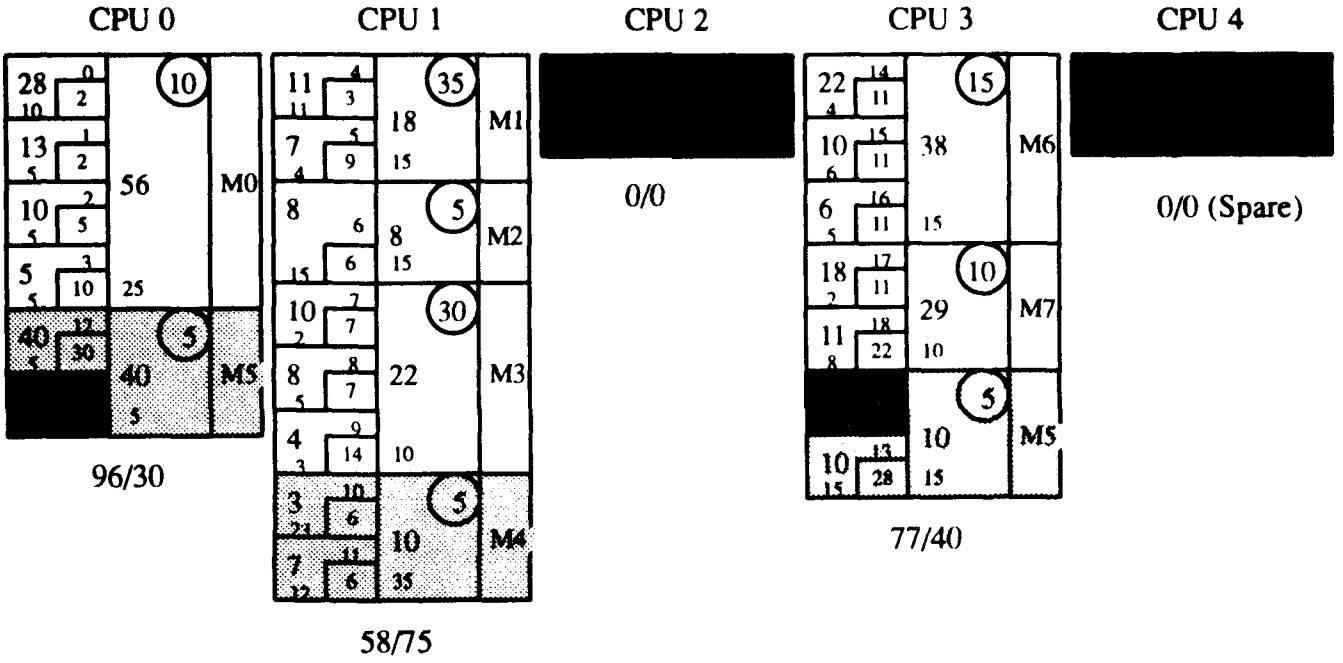


Figure 7 Reconfiguration Case 2: After

Test case 3, CPUs 1 and 4 are marked as failed, and the resulting configuration is shown in figure 8 on page 10.

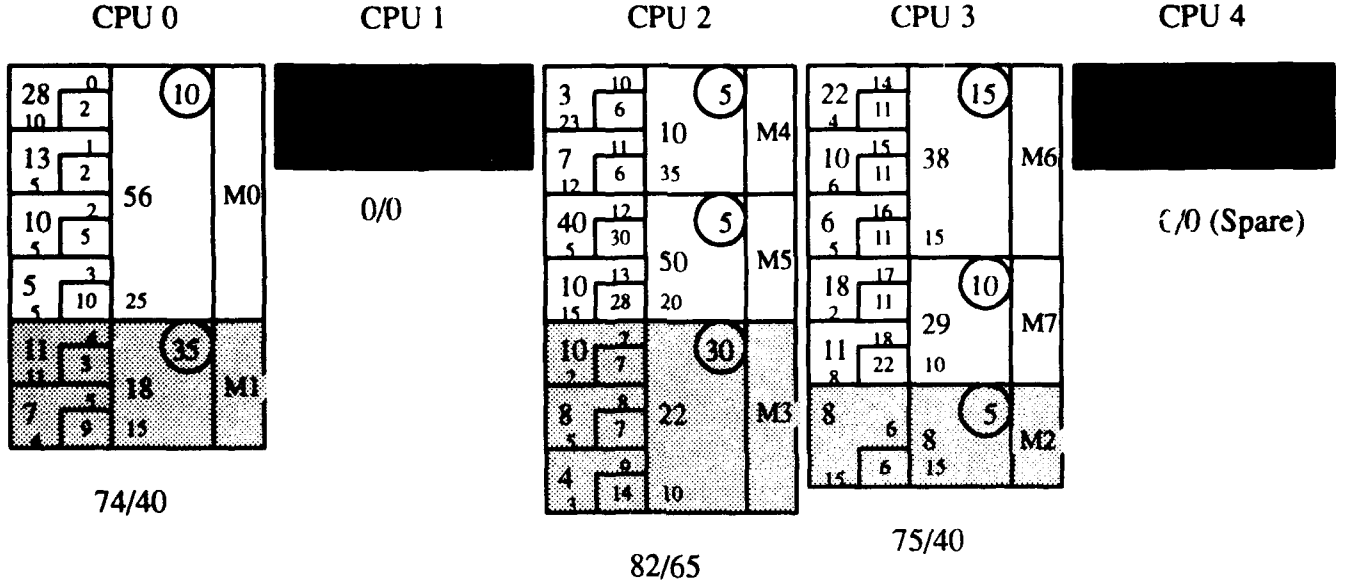


Figure 8 Reconfiguration Case 3: After

In addition to the mode/process definitions, each set of processes may have a communication graph depicting the source and destination of data flows. For the set of modes/processes depicted in figure 5, a sample communication graph is shown in figure 9 on page 11. In a complete communication graph, the directed arcs would be augmented with bandwidth data. Communication graphs are not required, but if a graph is

defined for a reconfiguration problem, it is checked against communication channel bandwidth limitations. When bandwidth limitations are exceeded, the clustering algorithm is used to check for another configuration which may exceed disturbance limitations, but not exceed channel bandwidth limits.

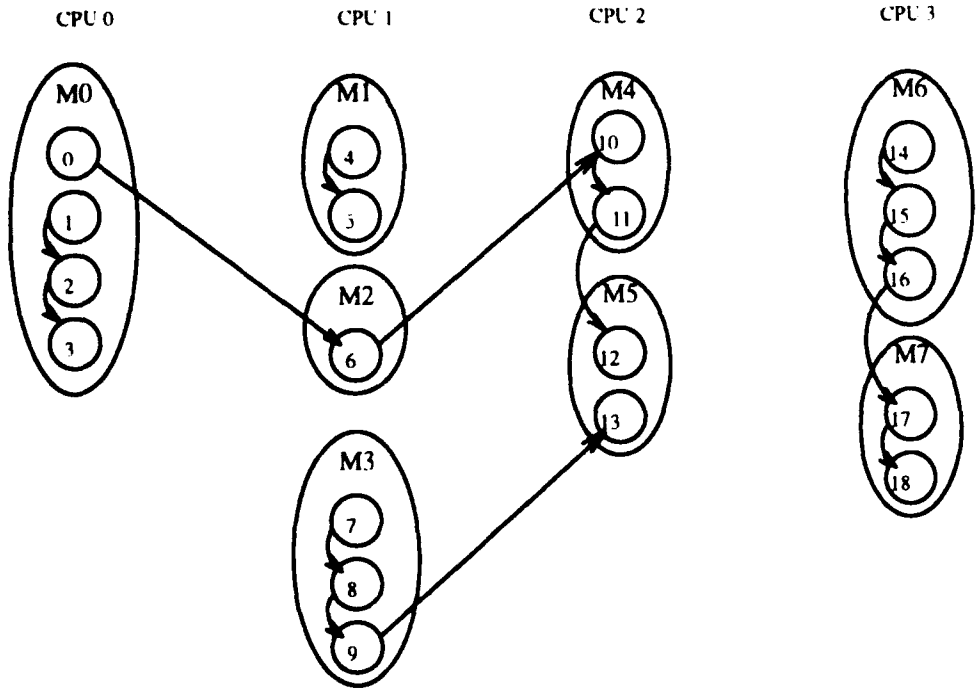


Figure 9 Sample Communication Graph

Test Case 4

The fourth test case demonstrates the case where greater than zero additional disturbance is required, and a mode from a non–failed CPU must be relocated. The before configuration is shown in figure 10 on page 12. The initial configuration is very similar to the initial configuration of the test cases above, only the percent CPU of process 0 is changed (from 28% to 33%).

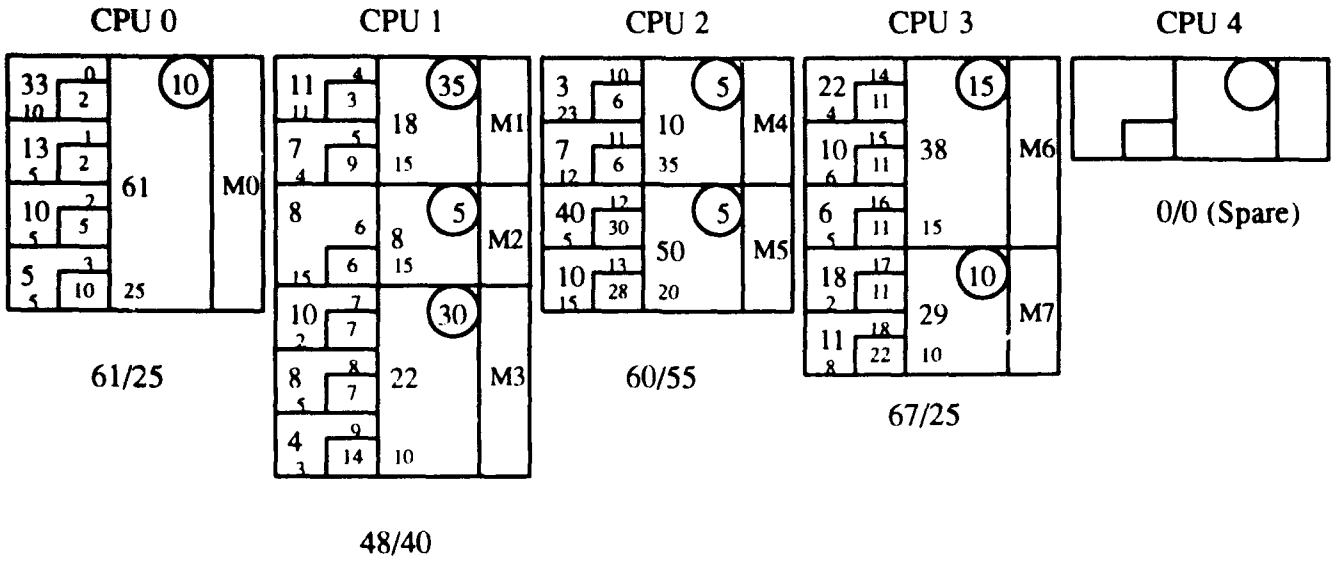


Figure 10 Test Case 4: Before

CPU 2 is marked as failed, and modes 4 and 5 must be relocated. This system cannot be reconfigured without incurring the additional disturbance cost of relocating a mode from a non–failed CPU. Mode 2 which started out in CPU 1 was moved to CPU3. Mode 4 from failed CPU 2 was moved to CPU 1, and mode 5 was split across CPUs 0 and 1. The results are shown in figure 11 on page 13.

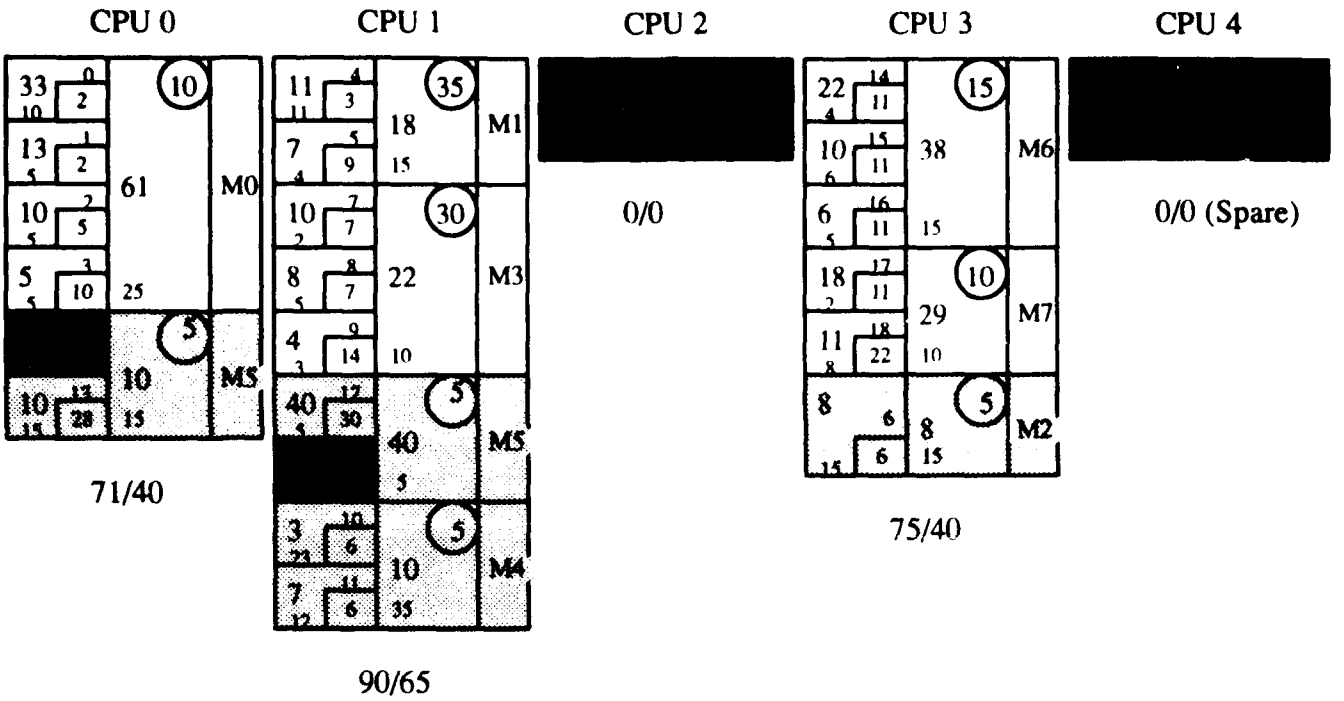


Figure 11 Test Case 4: After

Rate Monotonic Scheduling

Rate Monotonic Scheduling is a scheduling algorithm. When a set of tasks are being scheduled according to its rules, it is possible to determine the schedulability (whether or not all the tasks will meet their deadlines) by checking their periods and CPU utilization. RMS algorithm requires that the priority of each task be inversely proportional to its period. The theorem restated here is:

> *RMS1: A set of n independent preemptive periodic tasks scheduled by the rate monotonic algorithm will always meet its deadlines, for all task phasings, if:*

$$\frac{C_1}{T_1} + \ldots + \frac{C_n}{T_n} \leq U(n) = n(2^{\frac{1}{n}} - 1)$$

$U(1) = 1.0$ $U(4) = 0.756$ $U(7) = 0.728$
$U(2) = 0.828$ $U(5) = 0.734$ $U(8) = 0.724$
$U(3) = 0.779$ $U(6) = 0.734$ $U(9) = 0.720$

$U(\infty) = \ln 2 \approx 0.69$

Where:

C_i = Task Computation Time
T_i = Task Periodicity
U_i = Task Utilization

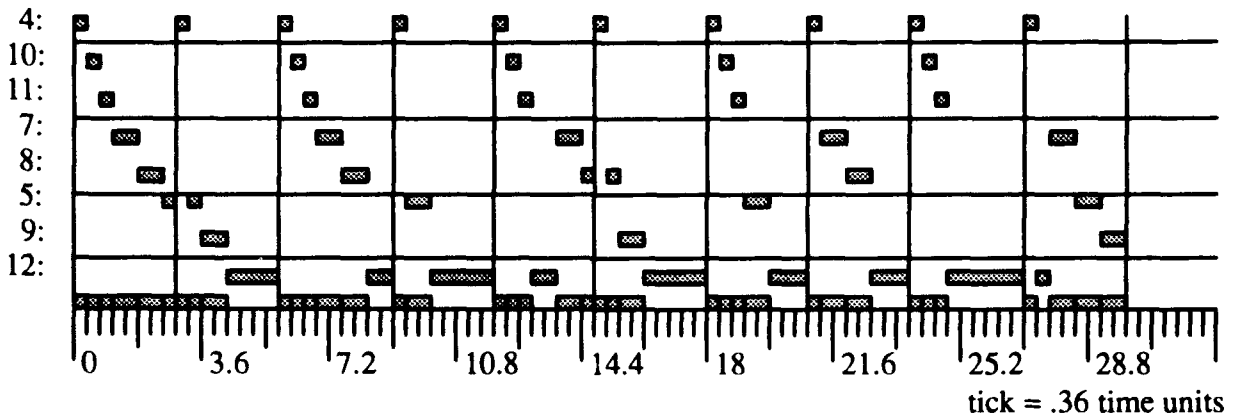
RMS1 is conservative, and if it doesn't show schedulability, RMS2 can be used for a more detailed test:

> *RMS2: For a set of independent, preemptive periodic tasks scheduled by RMS algorithm, start all tasks at t=0. If each meets its first deadline, the deadlines will always be met.*

Applying RMS1 to CPU 1, test case 2 after, results with:

Task 4: C/T = .11	.11 $\leq$ U(1) = 1.0
Task 10:C/T = .03	.14 $\leq$ U(2) = 0.828
Task 11:C/T = .07	.21 $\leq$ U(3) = 0.779
Task 7: C/T = .10	.31 $\leq$ U(4) = 0.756
Task 8: C/T = .08	.39 $\leq$ U(5) = 0.743
Task 5: C/T = .07	.46 $\leq$ U(6) = 0.734
Task 9: C/T = .04	.50 $\leq$ U(7) = 0.728
Task 12:C/T = .40	.90 $\geq$ U(8) = 0.724 Fails.

Since RMS1 fails to show that the tasks can successfully complete their processing by their deadlines, RMS2 can be used to perform a check with greater detail. Note that the longest period of any task in CPU 1 is 30, and applying RMS2 to test case 2, CPU 1:



.33 x 10 = 3.3
.18 x 5 = 0.9
.42 x 5 = 2.1
.70 x 5 = 3.5
.56 x 4 = 2.24
.69 x 4 = 2.76
.56 x 3 = 1.68

16.48

The first 7 tasks consume 16.48 time units of the 30 time unit frame. This leaves 13.52 time units for the last task, which only requires 12.00. The diagram cannot show all the blocks needed due to scaling and round off errors.

The RMS analysis performed above predicts that the set of processes will always meet their deadlines, and is a sample of the processing performed on the test bed during a reconfiguration calculation. Execution on the test bed demonstrated that the process set performed as expected in all test cases.

Conclusions

One interesting conclusion was a potential problem which may occur under the described reconfiguration algorithm set when future disturbance is considered. Excluding a configuration based on its future disturbance may exclude the only possible reconfiguration which satisfies all other constraints. This results in a false failure. A similar circumstance may occur when pseudo resources are strictly adhered to. A change is needed to allow a reconfiguration to occur even though its future disturbance score is too high or pseudo resource limit is passed, when that is the only reconfiguration where all the CPU and memory resources (real) are satisfied.

Using Rate Monotonic Scheduling to verify that the task set is schedulable and will meet all its deadlines has the potential to execute for longer than may be desirable. If only the conservative check is performed then execution time is bounded by the size of the task set. This alone is very efficient at verifying the schedulability of a set of tasks. But using only the conservative check leaves open the possibility of excluding a schedulable configuration. Performing the second check will prevent that possibility but at the cost of a potentially CPU intensive algorithm. The algorithm used in the test bed effectively identifies all the times when a task is started, stopped and preempted, for all tasks in a CPU for the period of the longest task in that CPU. The bound on its running time is O(NP) where N is the number of tasks and P is the greatest task period.

Using Rate Monotonic Scheduling has another drawback in that it applies only to sets of independent tasks. Tasks which pass data among themselves and spend time waiting for data are not independent. On the other hand tasks which communicate usually have periods which facilitate the exchange of data. This leads to a more constrained problem than the general problem of schedulability, possibly leading to simpler scheduling paradigms. The test cases in this report demonstrated that the test bed performed as expected when the tasks were independent.

Finally, the paradigms and algorithms used in the test bed need to be designed into a system from its beginning. Trying to convert an existing system to take advantage of them is likely to be difficult because of the operating environment needed. Task preemption, non–blocking system calls, control over the priority of tasks, and mechanisms for communicating configuration information to effected tasks all effect the architecture and design of a system and need to be considered before the system is implemented.

References

Advanced Distributed System Control Study — Reconfiguration Methodology, Final report for phase I: July 25 to December 31, 1983. Data item DS001 in IBM document 83–DB5–0007, for the contract Distributed System Reconfiguration Study, IBM Purchase Order number YD–289660.

Advanced Distributed System Control Study — Reconfiguration Methodology, Final report for phase I: July 25 to December 31, 1984. Deliverable technical report for the contract Distributed System Reconfiguration Study, IBM Purchase Order number 283261B–YD.

Ada Real–Time Programming: A Seminar, August 16, 1990. C. Douglass Locke, Thomas C. Ralya and David R. Vogel, IBM Federal Sector Division, Owego, NY 13827.

Integrated Systems, Inc.
3260 Jay Street
Santa Clara, California 95054
tel: (408) 980–1500
fax: (408) 980–0400

Appendices

A. Sample Report from the Test Bed System

The following is an excerpt from a report generated by executing test case 2 on the test bed system. At about the 65 time unit mark, CPU 2 was disabled. CPU 0 picked up process 12 and all the tasks started executing again around time unit 100. The test ran for approximately 200 time units before the report was generated.

```
Proc:t000**** CPU Id 0   Period: 2   Percnt:28   Loops:80   Misses: 0
 n   Start Finish Deadline Margin Missed Cstart Cstop
 0     211    267     411    144      0    211    267
 1     411    466     611    145      0    411    466
 2     611    666     811    145      0    611    666
 3     811    866    1011    145      0    811    866
 4    1011   1066    1211    145      0   1011   1066
 5    1211   1267    1411    144      0   1211   1267
 6    1411   1466    1611    145      0   1411   1466
 7    1611   1666    1811    145      0   1611   1666
 8    1811   1866    2011    145      0   1811   1866
 9    2011   2066    2211    145      0   2011   2066
10    2211   2266    2411    145      0   2211   2266
11    2411   2466    2611    145      0   2411   2466
12    2611   2666    2811    145      0   2611   2666
13    2811   2866    3011    145      0   2811   2866
14    3011   3066    3211    145      0   3011   3066
15    3211   3266    3411    145      0   3211   3266
16    3411   3466    3611    145      0   3411   3466
17    3611   3666    3811    145      0   3611   3666
18    3811   3866    4011    145      0   3811   3866
19    4011   4066    4211    145      0   4011   4066
20    4211   4266    4411    145      0   4211   4266
21    4411   4467    4611    144      0   4411   4467
22    4611   4666    4811    145      0   4611   4666
23    4811   4866    5011    145      0   4811   4866
24    5011   5066    5211    145      0   5011   5066
25    5211   5266    5411    145      0   5211   5266
26    5411   5467    5611    144      0   5411   5467
27    5611   5666    5811    145      0   5611   5666
28    5811   5866    6011    145      0   5811   5866
29    6011   6066    6211    145      0   6011   6066
30    6211   6266    6411    145      0   6211   6266
31    6411   6466    6611    145      0   6411   6466
32   10183  10238   10383    145      0  10183  10238
33   10383  10438   10583    145      0  10383  10438
34   10583  10638   10783    145      0  10583  10638
35   10783  10838   10983    145      0  10783  10838
36   10983  11038   11183    145      0  10983  11038
37   11183  11239   11383    144      0  11183  11239
38   11383  11438   11583    145      0  11383  11438
39   11583  11638   11783    145      0  11583  11638
40   11783  11838   11983    145      0  11783  11838
41   11983  12038   12183    145      0  11983  12038
42   12183  12239   12383    144      0  12183  12239
43   12383  12438   12583    145      0  12383  12438
```

44	12583	12638	12783	145	0	12583	12638
45	12783	12838	12983	145	0	12783	12838
46	12983	13038	13183	145	0	12983	13038
47	13183	13238	13383	145	0	13183	13238
48	13383	13438	13583	145	0	13383	13438
49	13583	13638	13783	145	0	13583	13638
50	13783	13838	13983	145	0	13783	13838
51	13983	14038	14183	145	0	13983	14038
52	14183	14238	14383	145	0	14183	14238
53	14383	14438	14583	145	0	14383	14438
54	14583	14638	14783	145	0	14583	14638
55	14783	14838	14983	145	0	14783	14838
56	14983	15038	15183	145	0	14983	15038
57	15183	15238	15383	145	0	15183	15238
58	15383	15439	15583	144	0	15383	15439
59	15583	15638	15783	145	0	15583	15638
60	15783	15838	15983	145	0	15783	15838
61	15983	16038	16183	145	0	15983	16038
62	16183	16238	16383	145	0	16183	16238
63	16383	16439	16583	144	0	16383	16439
64	16583	16638	16783	145	0	16583	16638
65	16783	16838	16983	145	0	16783	16838
66	16983	17038	17183	145	0	16983	17038
67	17183	17238	17383	145	0	17183	17238
68	17383	17438	17583	145	0	17383	17438
69	17583	17638	17783	145	0	17583	17638
70	17783	17838	17983	145	0	17783	17838
71	17983	18038	18183	145	0	17983	18038
72	18183	18238	18383	145	0	18183	18238
73	18383	18438	18583	145	0	18383	18438
74	18583	18638	18783	145	0	18583	18638
75	18783	18838	18983	145	0	18783	18838
76	18983	19038	19183	145	0	18983	19038
77	19183	19238	19383	145	0	19183	19238
78	19383	19438	19583	145	0	19383	19438
79	19583	19638	19783	145	0	19583	19638
80	0	0	19983	0	0	0	0

Proc:t001**** CPU Id 0 Period: 2 Percnt:13 Loops:80 Misses: 0

n	Start	Finish	Deadline	Margin	Missed	Cstart	Cstop
0	267	293	467	174	0	267	293
1	467	492	667	175	0	467	492
2	667	692	867	175	0	667	692
3	867	892	1067	175	0	867	892
4	1067	1092	1267	175	0	1067	1092
5	1267	1293	1467	174	0	1267	1293
6	1467	1492	1667	175	0	1467	1492
7	1667	1692	1867	175	0	1667	1692
8	1867	1892	2067	175	0	1867	1892
9	2067	2092	2267	175	0	2067	2092
10	2267	2292	2467	175	0	2267	2292
11	2467	2492	2667	175	0	2467	2492
12	2667	2692	2867	175	0	2667	2692
13	2867	2892	3067	175	0	2867	2892
14	3067	3092	3267	175	0	3067	3092
15	3267	3292	3467	175	0	3267	3292

16	3467	3492	3667	175	0	3467	3492
17	3667	3692	3867	175	0	3667	3692
18	3867	3892	4067	175	0	3867	3892
19	4067	4092	4267	175	0	4067	4092
20	4267	4292	4467	175	0	4267	4292
21	4467	4493	4667	174	0	4467	4493
22	4667	4692	4867	175	0	4667	4692
23	4867	4892	5067	175	0	4867	4892
24	5067	5092	5267	175	0	5067	5092
25	5267	5292	5467	175	0	5267	5292
26	5467	5493	5667	174	0	5467	5493
27	5667	5692	5867	175	0	5667	5692
28	5867	5893	6067	174	0	5867	5893
29	6067	6092	6267	175	0	6067	6092
30	6267	6292	6467	175	0	6267	6292
31	6467	6492	6667	175	0	6467	6492
32	10238	10264	10438	174	0	10238	10264
33	10438	10464	10638	174	0	10438	10464
34	10638	10664	10838	174	0	10638	10664
35	10838	10864	11038	174	0	10838	10864
36	11038	11064	11238	174	0	11038	11064
37	11239	11265	11438	173	0	11239	11265
38	11438	11464	11638	174	0	11438	11464
39	11638	11664	11838	174	0	11638	11664
40	11838	11864	12038	174	0	11838	11864
41	12038	12064	12238	174	0	12038	12064
42	12239	12264	12438	174	0	12239	12264
43	12438	12464	12638	174	0	12438	12464
44	12638	12664	12838	174	0	12638	12664
45	12838	12864	13038	174	0	12838	12864
46	13038	13064	13238	174	0	13038	13064
47	13238	13264	13438	174	0	13238	13264
48	13438	13464	13638	174	0	13438	13464
49	13638	13664	13838	174	0	13638	13664
50	13838	13864	14038	174	0	13838	13864
51	14038	14064	14238	174	0	14038	14064
52	14238	14264	14438	174	0	14238	14264
53	14438	14464	14638	174	0	14438	14464
54	14638	14664	14838	174	0	14638	14664
55	14838	14864	15038	174	0	14838	14864
56	15038	15064	15238	174	0	15038	15064
57	15238	15264	15438	174	0	15238	15264
58	15439	15465	15638	173	0	15439	15465
59	15638	15664	15838	174	0	15638	15664
60	15838	15864	16038	174	0	15838	15864
61	16038	16064	16238	174	0	16038	16064
62	16238	16264	16438	174	0	16238	16264
63	16439	16464	16638	174	0	16439	16464
64	16638	16664	16838	174	0	16638	16664
65	16838	16864	17038	174	0	16838	16864
66	17038	17064	17238	174	0	17038	17064
67	17238	17264	17438	174	0	17238	17264
68	17438	17464	17638	174	0	17438	17464
69	17638	17664	17838	174	0	17638	17664
70	17838	17864	18038	174	0	17838	17864
71	18038	18064	18238	174	0	18038	18064

```
72  18238  18264  18438    174      0  18238  18264
73  18438  18464  18638    174      0  18438  18464
74  18638  18664  18838    174      0  18638  18664
75  18838  18864  19038    174      0  18838  18864
76  19038  19064  19238    174      0  19038  19064
77  19238  19264  19438    174      0  19238  19264
78  19438  19464  19638    174      0  19438  19464
79  19638  19664  19838    174      0  19638  19664
80      0      0  20038      0      0      0      0

Proc:t002**** CPU Id 0   Period: 5   Percnt:10   Loops:32   Misses: 0
 n  Start Finish Deadline Margin Missed Cstart Cstop
 0    293    343    793    450      0    293    343
 1    793    924   1293    369      0    793    924
 2   1293   1343   1793    450      0   1293   1343
 3   1793   1924   2293    369      0   1793   1924
 4   2293   2343   2793    450      0   2293   2343
 5   2793   2924   3293    369      0   2793   2924
 6   3293   3342   3793    451      0   3293   3342
 7   3793   3924   4293    369      0   3793   3924
 8   4293   4342   4793    451      0   4293   4342
 9   4793   4924   5293    369      0   4793   4924
10   5293   5342   5793    451      0   5293   5342
11   5793   5924   6293    369      0   5793   5924
12   6293   6342   6793    451      0   6293   6342
13  10264  10314  10764    450      0  10264  10314
14  10764  10895  11264    369      0  10764  10895
15  11265  11314  11764    450      0  11265  11314
16  11764  11895  12264    369      0  11764  11895
17  12264  12314  12764    450      0  12264  12314
18  12764  12895  13264    369      0  12764  12895
19  13264  13314  13764    450      0  13264  13314
20  13764  13895  14264    369      0  13764  13895
21  14264  14314  14764    450      0  14264  14314
22  14764  14895  15264    369      0  14764  14895
23  15264  15314  15764    450      0  15264  15314
24  15764  15895  16264    369      0  15764  15895
25  16264  16314  16764    450      0  16264  16314
26  16764  16895  17264    369      0  16764  16895
27  17264  17314  17764    450      0  17264  17314
28  17764  17895  18264    369      0  17764  17895
29  18264  18314  18764    450      0  18264  18314
30  18764  18895  19264    369      0  18764  18895
31  19264  19314  19764    450      0  19264  19314
32      0      0  20264      0      0      0      0

Proc:t003**** CPU Id 0   Period:10   Percnt: 5   Loops:17   Misses: 0
 n  Start Finish Deadline Margin Missed Cstart Cstop
 0    343    393   1343    950      0    343    393
 1   1343   1393   2343    950      0   1343   1393
 2   2343   2393   3343    950      0   2343   2393
 3   3343   3393   4343    950      0   3343   3393
 4   4343   4392   5343    951      0   4343   4392
 5   5343   5392   6343    951      0   5343   5392
 6   6343   6392   7343    951      0   6343   6392
 7  10314  10364  11314    950      0  10314  10364
```

```
 8  11314  11364  12314    950      0  11314  11364
 9  12314  12364  13314    950      0  12314  12364
10  13314  13364  14314    950      0  13314  13364
11  14314  14364  15314    950      0  14314  14364
12  15314  15364  16314    950      0  15314  15364
13  16314  16364  17314    950      0  16314  16364
14  17314  17364  18314    950      0  17314  17364
15  18314  18364  19314    950      0  18314  18364
16  19314  19364  20314    950      0  19314  19364
17      0      0  21314      0      0      0      0

Proc:t012**** CPU Id 0   Period:30   Percnt:40   Loops: 3   Misses: 0
 n  Start Finish Deadline Margin Missed Cstart Cstop
 0  10364  12968  13364    396      0  10364  12968
 1  13364  15969  16364    395      0  13364  15969
 2  16364  18969  19364    395      0  16364  18969
 3  19364      0  22364      0      0  19364      0
```

Loops indicates the number of cycles executed by the model process.

Missed is the number of missed deadlines.

n is the cycle index.

Start, Finish, and Deadline are points in time. All time figures are in 100th's of a time unit (for example, process t000 finished cycle 1 at 4.66 time units from the start of the test).

Margin is the amount of time between **Finish** and **Deadline**.

Missed is a 0/1 missed deadline indicator.

Cstart and **Cstop** are time tags for the start and completion of the CPU busy loop of the model process.

This report was prepared under Interleaf 5.2. A template for the ANSI Z39.18 standard was created and used and is available upon request.